AFRL-RV-PS-
TR-2014-0168

AFRL-RV-PS-
TR-2014-0168

DOSIMETER DESIGN PROGRAM

Craig J. Kief

**COSMIAC at UNM
2350 Alamo Avenue SE, Ste 300
Albuquerque, NM 87106**

5 Jan 2015

Final Report

**AIR FORCE RESEARCH LABORATORY
Space Vehicles Directorate
3550 Aberdeen Ave SE
AIR FORCE MATERIEL COMMAND
KIRTLAND AIR FORCE BASE, NM 87117-5776**

DTIC COPY
NOTICE AND SIGNATURE PAGE

AFRL-RV-PS-TR-2014-0168 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

//SIGNED//
KEITH AVERY
Program Manager

//SIGNED//
JAMES LYKE
Tech Advisor, Space Electronics Protection Branch

//SIGNED//

BENJAMIN M. COOK, Lt Col, USAF
Deputy Chief, Spacecraft Technology Division
Space Vehicles Directorate

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. REPORT DATE *(DD-MM-YY)* 05-01-2015	**2. REPORT TYPE** Final Report	**3. DATES COVERED** *(From - To)* 16 Jul 2013 to 16 Oct 2014

4. TITLE AND SUBTITLE Dosimeter Design Program	**5a. CONTRACT NUMBER** FA9453-13-1-0283
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER 63401F
6. AUTHOR(S) Craig J. Kief	**5d. PROJECT NUMBER** 2181
	5e. TASK NUMBER PPM00017001
	5f. WORK UNIT NUMBER EF011499
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) COSMIAC at UNM, 2350 Alamo Avenue SE, Ste 300, Albuquerque, NM 87106	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory, Space Vehicles Directorate, 3550 Aberdeen Ave., SE, Kirtland AFB, NM 87117-5776	**10. SPONSOR/MONITOR'S ACRONYM(S)** AFRL/RVSE
	11. SPONSOR/MONITOR'S REPORT NUMBER(S) AFRL-RV-PS-TR-2014-0168

12. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for Public Release; Distribution is Unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
The purpose of this research project was to explore the various aspects of dosimeter development with a particular focus on nanosatellites. The team at the University of New Mexico's COSMIAC Center created a series of two different dosimeters for space flight. The first dosimeter was for low earth orbit and as such, didn't require radiation hardened electronics. The second dosimeter was created to fly in a much higher environment so it was designed to operate with radiation hardened electronics. The developed dosimeter system is called the Radiation Hazard Assessment System (RHAS). The RHAS was developed to monitor the radiation environment in a geosynchronous satellite.

15. SUBJECT TERMS
Radiation testing, Cobalt, Microcontroller

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Keith Avery
a. REPORT Unclassified	**b. ABSTRACT** Unclassified	**c. THIS PAGE** Unclassified	Unlimited	78	**19b. TELEPHONE NUMBER** *(include area code)*

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. 239.18

(This page intentionally left blank)

ACKNOWLEDGEMENT

This material is based on research sponsored by Air Force Research Laboratory or the U.S. Government under agreement number FA9453-13-1-0283. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon."

DISCLAIMER

Table of Contents

1. Summary

The University of New Mexico (UNM) proposed to develop a series of dosimeter flight articles for use in future small satellite missions. The Configurable Space Microsystems Innovations and Applications Center (COSMIAC) aerospace and defense center at UNM worked on two CubeSats for delivery to the National Aeronautics and Space Administration (NASA) in 2013 and 2014. Dosimeters provide a critical way to measure the radiation at various altitudes and orbital inclinations. By understanding the actual levels, this allows developers of large (and very expensive) satellites to be able to use less expensive parts that meet minimum requirements for survivability.

2. Introduction

The current trend in nanosatellites is the use of a standard called the CubeSat. The CubeSat form factor is described with the use of "U" for units. A 1U is 10cm x 10cm x 10cm. A 6U is 10cm x 20cm x 30cm. UNM completed the design build and launch of a 1U CubeSat called Trailblazer (see Figure 1) that was launched in November of 2013 upon the ORS-3 mission. In addition, COSMIAC is working to complete a 6U spacecraft called ORS Squared for launch in late 2014. This spacecraft is shown in Figure 2.

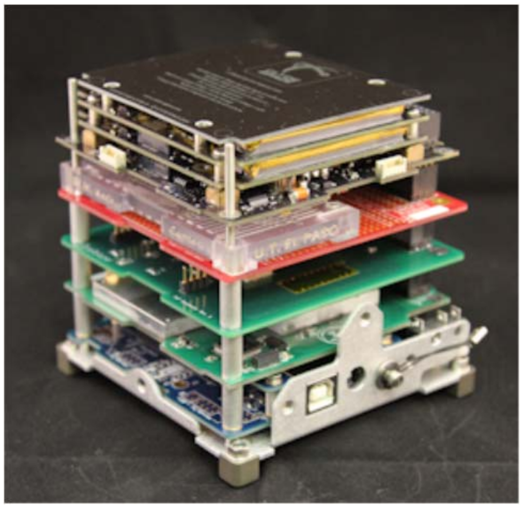

Figure 1. Trailblazer

The team said that they would use the provided funds to complete three major goals:

- Design of a dosimeter that is not radiation hardened. The team will work to create a dosimeter design using parts that are not radiation hardened. This will allow for a dosimeter that can be used on any inexpensive CubeSat designed for

Low Earth Orbit (LEO). Satellites flown in LEO are often provided with natural protection from harmful effects normally found in the outer environment. The Earth's natural barriers provide protection for the planets occupants as well as spacecraft that stay close to the Earth's surface (200km – 400km) from many of the effects of solar radiation. This means that standard commercial electronics (which are much cheaper) can often be utilized. Non-hardened parts can often be an order of magnitude less in cost.

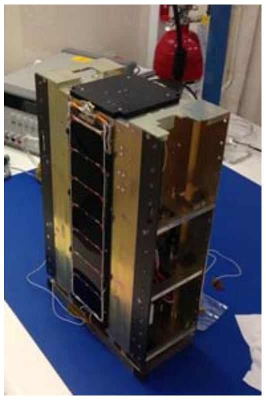

Figure 2. ORS

- Design a dosimeter that is radiation hardened. We will use the funding to create initial designs for a dosimeter that is built on radiation hardened parts designed for much higher orbits. There will be no prototype developed under this activity since the cost for the radiation hardened parts would be prohibitive. Various chips that are radiation hardened will be purchased for analysis.
- Develop Space Plug-and-play Architecture (SPA) interfaces for the dosimeters. The Air Force Research Laboratory has created a bus architecture call the SPA. This design paradigm allows for easy integration of modules into a spacecraft or other system. The team will work to develop a pathway ahead to make sure that both versions of the dosimeter could have a SPA interface.

3. Methods, Assumptions, and Procedures

Goal 1: The team created a simple non hardened dosimeter that was completed in time to be flown on the Trailblazer satellite in November of 2013. Unfortunately, this satellite was never heard from once on orbit so minimum results from on orbit are available. However, this design was also evaluated for flight in future COSMIAC satellites. An example schematic for the board is shown in Figure 3 (full schematics are provided upon request). We believe there is significant value to this design in its simplicity. We began work with other schools within the AFRL University Nanosat Program (UNP) to see if this is a sensor that they would like to fly on their spacecraft. Its design (with its low power and inexpensive parts) makes it an excellent sensor platform for a UNP satellite. We can provide any UNP partner with the complete schematics, bill of materials and board design files.

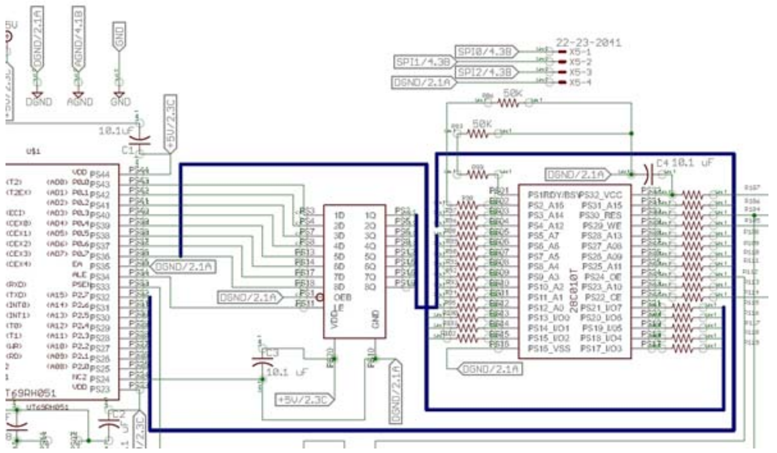

Figure 3. Board schematic subset

Goal 2: The radiation hardened version of this dosimeter board has been designed. Although there was no initial intent to actually create the hardware platform, resources and unforeseen opportunities provided the ability for the team to actually create this type of platform. We created the board as shown in Figure 4. COSMIAC has coordinated to find a flight opportunity for this dosimeter board on a Massachusetts Institute of Technology (MIT)/Lincoln Lab satellite for 2015. Engineering models were created and provided to integration teams at MIT. All problems that have been identified so far have been resolved. The schematics for the board are very similar to the ones shown in Figure 3 for the non-hardened version. The main difference is the use of parts that can withstand the rigors of deep space. With the assistance of AFRL's Space Weather Center of Excellence, we were able to do flight environmental testing and qualification. Electromagnetic interference testing identified potential areas of concern that were addressed by the inclusion of increased metal sheeting. AFRL Space Vehicles personnel assisted with total dose radiation testing on Kirtland Air Force Base. The developed dosimeter system was called the Radiation Hazard Assessment System (RHAS). The RHAS was originally developed to monitor the radiation environment in a geosynchronous satellite. It incorporates three Teledyne µDos001 dosimeters to measure total ionizing dose resulting from either proton or electron strikes in the space environment. The dosimeters employ a silicon detector which responds to energy deposits in the range of 100 kilo-electron volt to 15 Mega-electron volt. The current pulses from the detector are integrated, and when the resulting charge quanta reach a threshold equivalent to 14 micro-radians, a counter is incremented. The counter output is applied to a digital to analog converter which drives the three output ranges (low, medium, and high) that are progressively scaled by 256 (i.e., medium range = 256*low range, etc.).

The RHAS instrument shown in Figure 4 was designed to fit in a CubeSat format. The three dosimeters are located on the top side of the instrument board. In their current application, a different effective thickness of shielding (not shown here) is applied over each dosimeter to provide an indication of the energy spectrum of the space radiation. The instrument incorporates radiation hardened electronics including: an Aeroflex 8051 microcontroller, a Maxwell Electrically Erasable Programmable Read-Only Memory (EEPROM), Texas Instrument analog to digital converters, and associated support electronics. Appendix A contains the source code used for this instrument. The electronics are specified to a hardness level of 1 Mega-radian/Silicon total dose and are latch-up free. COSMIAC can provide additional radiation testing to higher total dose levels if required.

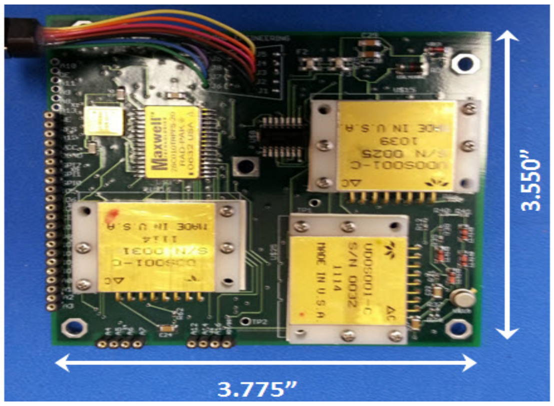

Figure 4. Flight model of RadHard Dosimeter

Goal 3: The tricky part of SPA is that the versions changed in the middle of the project. The original version that existed when this proposal was written was called the Satellite Data Module (SDM). The new version is called the Satellite System Module (SSM). SSM is much more mature and of higher quality. Also, this software has now been released as open source by AFRL. This greatly increases the ability to utilize it for student projects. The non-hardened version of the dosimeter board is already SPA compatible (older SDM version). The development of the SPA version of the hardened board was evaluated and then decided against. Future plans for SPA within the larger satellite community are very much in a state of flux so it was deemed that doing this development (at this time) would provide very limited benefit. Evaluation was made on what needed to be accomplished should this be desired later. Hardware and software stubs have been put into place to allow for easy integration at a later time.

4. Results and Discussion

The results to date are that the first iterations of the boards have been created. Ongoing testing has begun but will need to be continued for months to completely confirm that the systems are operating within acceptable parameters. Since a launch opportunity has been found, the next iterations will involve the creation of a control system to be able to interface the dosimeter into a larger orbital platform.

5. Conclusions

The conclusions are that it is possible for an academic institution to be able to create a radiation detection system on a limited budget that has space community level utility to achieve the required work for measurements in a space environment.

Appendix: Source Files for RHAS

File List

Here is a list of all files with brief descriptions:

File Documentation

ADC.c File Reference

#include "main.h"

Functions

- tByte **read_dosimeter_all** (tByte mode2)
- tByte **read_dosimeter_low** ()
- tByte **check_rollover** (tByte val, tByte old_val)
- tWord **compute_avg** (tWord x, tWord xmean)

Variables

- tByte **tick** = 0
- tByte **mode**
- tByte **count** = 0
- tWord **curr_val**
- tByte **k** [] = {0, 0, 0, 0, 0}
- tLong idata **dos** [3]
  INSTRUMENT CONTROL Dosimeter Telemetry Application (Main module) (Version 2.0: Flight Software)
- tLong idata **dos_zero** [3]
- tLong idata **cal_pre** [3]
- tLong idata **cal_post** [3]
- tLong idata **TickCount**
- Status1_type idata **Status1**
- tByte code **adc_ch** [20]
- tByte code **adc_ch_mode2** [3][5]
- tWord idata **ps_15V**
- tWord idata **ps_5V**
- tWord idata **temps** [3]
- bit **RunOnce**
- bit **ReadAllIsRunning**

Function Documentation

tByte check_rollover (tByte val, tByte old_val)

Definition at line 325 of file ADC.c.

tWord compute_avg (tWord x, tWord xmean)

Definition at line 338 of file ADC.c.

tByte read_dosimeter_all (tByte mode2)

Definition at line 45 of file ADC.c.

tByte read_dosimeter_low ()

Definition at line 227 of file ADC.c.

Variable Documentation

tByte code adc_ch[20]

Definition at line 104 of file MAIN.C.

tByte code adc_ch_mode2[3][5]

Definition at line 91 of file MAIN.C.

tLong idata cal_post[3]

Definition at line 36 of file MAIN.C.

tLong idata cal_pre[3]

Definition at line 35 of file MAIN.C.

tByte count = 0

Definition at line 8 of file ADC.c.

tWord curr_val

Definition at line 9 of file ADC.c.

tLong idata dos[3]

INSTRUMENT CONTROL Dosimeter Telemetry Application (Main module) (Version 2.0: Flight Software)

This code implements the full INSTRUMENT CONTROL dosimeter sensor, including all message formatting, data processing, and utility functions (ADC, CRC, ...)

Version Rev. 1.0 Created from the code written by Brian Zufelt for the development board and by code written by me for the EU model.

Chip type : Aeroflex UTMC UT69RH051 Program type : Firmware Core Clock frequency : 16.129000 MHz

Definition at line 33 of file MAIN.C.

tLong idata dos_zero[3]

Definition at line 34 of file MAIN.C.

tByte k[] = {0, 0, 0, 0, 0}

Definition at line 10 of file ADC.c.

tByte mode

Definition at line 7 of file ADC.c.

tWord idata ps_15V

Definition at line 38 of file MAIN.C.

tWord idata ps_5V

Definition at line 39 of file MAIN.C.

bit ReadAllIsRunning

Definition at line 81 of file MAIN.C.

bit RunOnce

Definition at line 84 of file MAIN.C.

Status1_type idata Status1

Definition at line 78 of file MAIN.C.

tWord idata temps[3]

Definition at line 40 of file MAIN.C.

tByte tick = 0

Definition at line 6 of file ADC.c.

tLong idata TickCount

Definition at line 57 of file MAIN.C.

```
// ADC.c
// This code implements all variants of the ADC functions

#include "main.h"

tByte tick = 0;
tByte mode;
tByte count = 0;
tWord curr_val;
tByte k[] = {0, 0, 0, 0, 0};

extern tLong idata dos[3];
extern tLong idata dos_zero[3];
extern tLong idata cal_pre[3];
extern tLong idata cal_post[3];
extern tLong idata TickCount;
extern Status1_type idata Status1;
extern tByte code adc_ch[20];
extern tByte code adc_ch_mode2[3][5];
extern tWord idata ps_15V;
extern tWord idata ps_5V;
extern tWord idata temps[3];
extern bit RunOnce;
extern bit ReadAllIsRunning;

/*******************************************************************
* read_dosimeter_all()
* Reads the actual values of the high, medium, and low channels
* of a dosimeter and stores the results in a 32-bit value (in the
* form:  High:12, Med:12, Low:8
*
* This is done anytime we want the actual values (rather than the
* accumulated dose over a 2-minute period).
*
* Inputs:
*   Formal name   Type                    Purpose
*
* Return value:
*   Type       Purpose
*   tByte       0 = Success
*             -1 = Error: NULL pointer
*             -2 = Error: Invalid Dosimeter number
********************************************************************/

tByte read_dosimeter_all( tByte mode2 )
```

```
{
  tWord l = 0;
  tWord m = 0;
  tWord h = 0;
  tByte i = 0;

  ReadAllIsRunning = 1;

  if ((RunOnce == 0)) {
    // Do this once at startup and each mode change
    RunOnce = 1;
    count = 0;
    tick = 0;
  }

  // Burn one ADC operation for each device to set up read channel
  sample_adc_fast2(0x00);  // Set the first ADC0 chan to read DOS1_LOW
  sample_adc_fast2(0x15);  // Set the first ADC1 chan to read DOS3_LOW

  for (i = 0; i < 16; i++) {
    l += sample_adc_fast2( 0x00 );
  }
  sample_adc_fast2( 0x01 );
  for (i = 0; i < 16; i++) {
    m += sample_adc_fast2( 0x01 );
  }
  sample_adc_fast2( 0x02 );
  for (i = 0; i < 16; i++) {
    h += sample_adc_fast2( 0x02 );
  }
  sample_adc_fast2( 0x03 );
  m = (m >> 4);  // Don't shift h and l

#ifdef __DEBUG__
  l = m = h = 0;
#endif  // __DEBUG__

  // The following (commented out) is what we want to accomplish
  // However, this method involves several long functions calls and
  // requires 454 uS to complete.
  //dos[0] = ((tLong )h << 20) | ((tLong )m << 8) | ((tLong )l >> 4);
  // However, There is not time during the interrupt service routine
  // to perform all 9 ADC operations and this calculation (times three).

  // This is a hack that accomplishes the same result and only takes
  // 30.5 uS to complete!!  This is the magic of hand coding.

  // The next two lines are commented out because they are offset by
  // the collection of 16 samples.
  // h = h << 4;
  // l = l << 4;

*(tByte *)(dos) = *(tByte *)(&h);
  *((tByte *)dos+1) = (*(((tByte *)&h)+1) & 0xF0) | (*(tByte *)(&m) & 0x0F);
  *((tByte *)dos+2) = *(((tByte *)&m)+1);
  *((tByte *)dos+3) = *(tByte *)(&l);

  // Zero local values
  l = m = h = 0;

  // Read DOS2_LOW sixteen times
  for (i = 0; i < 16; i++) {
    l += sample_adc_fast2( 0x03 );
  }
  sample_adc_fast2( 0x04 );

  // Read DOS2_MID sixteen times
  for (i = 0; i < 16; i++) {
    m += sample_adc_fast2( 0x04 );
  }
  sample_adc_fast2( 0x05 );

  // Read DOS2_HI sixteen times
  for (i = 0; i < 16; i++) {
    h += sample_adc_fast2( 0x05 );
  }
  m = (m >> 4);  // Don't shift h and l

  // If we are in Analysis Mode 2, we need to read subcommutated values
  if (mode2 == 1) {
    sample_adc_fast2( adc_ch_mode2[0][count] );   // "Decision Stage 0"
  } else {
    sample_adc_fast2( 0x00 );
  }

#ifdef __DEBUG__
  h = m = l = 0;
#endif  // __DEBUG__

  *(tByte *)(dos+4) = *(tByte *)(&h);
  *((tByte *)dos+5) = (*(((tByte *)&h)+1) & 0xF0) | (*(tByte *)(&m) & 0x0F);
  *((tByte *)dos+6) = *(((tByte *)&m)+1);
  *((tByte *)dos+7) = *(tByte *)(&l);

  // Zero local values
  l = m = h = 0;
  for (i = 0; i < 16; i++) {
    l += sample_adc_fast2( 0x15 );
  }
  sample_adc_fast2( 0x16 );
  for (i = 0; i < 16; i++) {
    m += sample_adc_fast2( 0x16 );
  }
  sample_adc_fast2( 0x17 );
  for (i = 0; i < 16; i++) {
    h += sample_adc_fast2( 0x17 );
  }
  m = (m >> 4);  // Don't shift h and l
  if (mode2 == 1) {
    sample_adc_fast2( adc_ch_mode2[1][count] );  // "Decision Stage 1"
  } else {
    sample_adc_fast2( 0x15 );
  }

  #ifdef __DEBUG__
  l = m  = h = 0;
#endif  // __DEBUG__

  *(tByte *)(dos+8) = *(tByte *)(&h);
  *((tByte *)dos+9) = (*(((tByte *)&h)+1) & 0xF0) | (*(tByte *)(&m) & 0x0F);
  *((tByte *)dos+10) = *(((tByte *)&m)+1);
  *((tByte *)dos+11) = *(tByte *)(&l);

  // Perform one "extra" ADC per tick of the System Timer
  // Truncate the current value to 8 bits
  if ( mode2 == 1) {
    curr_val = sample_adc_fast2( adc_ch_mode2[2][count] );   // "Decision Stage 2"
  }

#ifdef __DEBUG__
  curr_val = 0x0A5A;
#endif // __DEBUG__

  if (mode2 == 1)
  {
    // Read the next 'miscellaneous' channel
    switch (count) {
      case 0:
        ps_15V = compute_avg( curr_val, ps_15V );
        break;
      case 1:
        ps_5V = compute_avg( curr_val, ps_5V );
        break;
      case 2:
        temps[0] = compute_avg( curr_val, temps[0] );
        break;
      case 3:
        temps[1] = compute_avg( curr_val, temps[1] );
        break;
      case 4:
        temps[2] = compute_avg( curr_val, temps[2] );
        break;
      default:
        ;
    }
    count++;
    // Count goes: 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, ...
    if (count == 5) count = 0;  // May be superfluous
  }

  ReadAllIsRunning = 0;

  // Return no error
  return 0;
}

/*******************************************************************
* read_dosimeter_low()
* Reads only the value of the low channel of all dosimeters, truncates
* the value into an 8 bit value, checks for a rollover condition, and
* adds the current (truncated) value to the 32-bit value (in the
* form:  High:12, Med:12, Low:8
*
* Inputs:
*   None
*
* Return value:
*   Type       Purpose
*   tByte       0 = Success
*******************************************************************/
```

```
tByte read_dosimeter_low( )
{

#ifdef __DEBUG__
  #define DUMMY_INC1 321
  #define DUMMY_INC2 20
  #define DUMMY_INC3 14
  static tWord dummy_val1 = 0;
  static tWord dummy_val2 = 0;
  static tWord dummy_val3 = 0;
#endif // __DEBUG__

  if (RunOnce == 0) {
    // Do this once at startup and each mode change
    RunOnce = 1;
    count = 0;
    tick = 0;
    sample_adc_fast2(0x00);  // Set the first ADC0 chan for present mode
    sample_adc_fast2(0x15);  // Set the first ADC1 chan for present mode
  }

  curr_val = sample_adc_fast2( adc_ch[count++] );

#ifdef __DEBUG__
    dummy_val1 = (dummy_val1 + DUMMY_INC1) & (tWord )0x0FFF;
    curr_val = dummy_val1;
#endif  // __DEBUG__

  curr_val = curr_val >> 4;
  if (check_rollover((tByte )curr_val, (tByte)dos[0])) {
    dos[0] = ( (dos[0] & 0xFFFFFF00) | (tLong )curr_val) + 256;
  } else {
    dos[0] = ( (dos[0] & 0xFFFFFF00) | (tLong )curr_val);
  }

  curr_val = sample_adc_fast2( adc_ch[count++]);

  #ifdef __DEBUG__
    dummy_val2 = (dummy_val2 + DUMMY_INC2) & (tWord )0x0FFF;
    curr_val = dummy_val2;
#endif  // __DEBUG__

  curr_val = curr_val >> 4;
  if (check_rollover((tByte )curr_val, (tByte)dos[1])) {
    dos[1] = ( (dos[1] & 0xFFFFFF00) | (tLong )curr_val) + 256;
  } else {
    dos[1] = ( (dos[1] & 0xFFFFFF00) | (tLong )curr_val);
  }

  curr_val = sample_adc_fast2( adc_ch[count++] );

#ifdef __DEBUG__
    dummy_val3 = (dummy_val3 + DUMMY_INC3) & (tWord )0x0FFF;
    curr_val = dummy_val3;
#endif  // __DEBUG__

  curr_val = curr_val >> 4;
  if (check_rollover((tByte )curr_val, (tByte)dos[2])) {
    dos[2] = ( (dos[2] & 0xFFFFFF00) | (tLong )curr_val) + 256;
  } else {
    dos[2] = ( (dos[2] & 0xFFFFFF00) | (tLong )curr_val);
  }

  // Truncate the current value to 8 bits
  curr_val = sample_adc_fast2( adc_ch[count++] );

#ifdef __DEBUG__
  curr_val = 0x0A5A;
#endif // __DEBUG__

  // Read the next 'miscellaneous' channel
  switch (tick) {
    case 0:
      ps_15V = compute_avg( curr_val, ps_15V );
      break;
    case 1:
      ps_5V = compute_avg( curr_val, ps_5V );
      break;
    case 2:
      temps[0] = compute_avg( curr_val, temps[0] );
      break;
    case 3:
      temps[1] = compute_avg( curr_val, temps[1] );
      break;
    case 4:
      temps[2] = compute_avg( curr_val, temps[2] );
      break;
    default:
      //tick = 0;
    ;
  }
  tick++; if (tick == 5) tick = 0;
  if (count == 20) count = 0;

  // Return no error
  return 0;
}

tByte check_rollover( tByte val, tByte old_val )
{
  /*  Rollover condition:
      -------------------------
        Old value:  11xx xxxx
        New value:  00xx xxxx
  */
  if ( ((old_val & 0xC0) == 0xC0) && ((val & 0xC0) == 0x00) )
    return 1; // Rollover detected
  else
    return 0;
}

tWord compute_avg( tWord x, tWord xmean )
{
  tWord xtemp;

  if (k[tick] == 0) {
    k[tick] = 1;
    xtemp = x << 4;
  } else {
    xtemp  = xmean >> 1;  // xtemp = 1/2 * xmean
    xtemp += xmean >> 2;  // xtemp = 3/4 * xmean
    xtemp += xmean >> 3;  // xtemp = 7/8 * xmean
    xtemp += xmean >> 4;  // xtemp = 15/16 * xmean
    xtemp += x;           // xtemp = (15/16 * xmean) + x
  }
  return xtemp;
}
```

Dosimeter.c File Reference

#include "main.h"

Macros

- #define **CAL_PULSE_WIDTH** 17

Functions

- void **Init_INSTRUMENT CONTROL** (void)
- void **SysTimerSetupFast** (void)
- void **SysTimerSetupSlow** (void)
- void **SystemTick** (void)
- void **TSK_DoCal** ()
- void **initialize_normal_mode** (void)
- void **initialize_startup_mode** (void)
- void **ChangeToAnalysisMode2** (void)
- void **ChangeToAnalysisMode1** (void)
- void **TSK_DoADC** (void)

Variables

- Status1_type idata **Status1**
- Status2_type idata **Status2**
- MCF_type idata **MCF**
- tLong idata **dos** [3]

  INSTRUMENT CONTROL Dosimeter Telemetry Application (Main module) (Version 2.0: Flight Software)
- tLong idata **dos_zero** [3]
- tLong idata **cal_pre** [3]
- tLong idata **cal_post** [3]
- tWord idata **dos_livetime** [3]
- tLong idata **TickCount**
- tLong idata **TotalTickCount**
- tByte idata **HourCount**
- bit **TelemetryTick**
- bit **ADC_Start**
- bit **CalRunning**
- bit **ReadAllIsRunning**
- bit **CalMode**
- bit **RunOnce**
- bit **GSE**
- bit **INSTRUMENT CONTROL_Enable**
- bit **Dos1TurnOn**
- bit **Dos2TurnOn**
- bit **Dos3TurnOn**

- tLong idata **RxTimer**

Macro Definition Documentation

#define CAL_PULSE_WIDTH 17

Definition at line 5 of file Dosimeter.c.

Function Documentation

void ChangeToAnalysisMode1 (void)

Definition at line 279 of file Dosimeter.c.

void ChangeToAnalysisMode2 (void)

Definition at line 269 of file Dosimeter.c.

void Init_INSTRUMENT CONTROL (void)

Definition at line 31 of file Dosimeter.c.

void initialize_normal_mode (void)

Definition at line 210 of file Dosimeter.c.

void initialize_startup_mode (void)

Definition at line 239 of file Dosimeter.c.

void SystemTick (void)

Definition at line 113 of file Dosimeter.c.

void SysTimerSetupFast (void)

Definition at line 57 of file Dosimeter.c.

void SysTimerSetupSlow (void)

Definition at line 85 of file Dosimeter.c.

void TSK_DoADC (void)

Definition at line 289 of file Dosimeter.c.

void TSK_DoCal ()

Definition at line 162 of file Dosimeter.c.

Variable Documentation

bit ADC_Start

Definition at line 47 of file MAIN.C.

tLong idata cal_post[3]

Definition at line 36 of file MAIN.C.

tLong idata cal_pre[3]

Definition at line 35 of file MAIN.C.

bit CalMode

Definition at line 54 of file MAIN.C.

bit CalRunning

Definition at line 53 of file MAIN.C.

tLong idata dos[3]

INSTRUMENT CONTROL Dosimeter Telemetry Application (Main module) (Version 2.0: Flight Software)

This code implements the full INSTRUMENT CONTROL dosimeter sensor, including all message formatting, data processing, and utility functions (ADC, CRC, ...)

Version Rev. 1.0 Created from the code written by Brian Zufelt for the development board and by code written by me for the EU model.

Chip type : Aeroflex UTMC UT69RH051 Program type : Firmware Core Clock frequency : 16.129000 MHz

Definition at line 33 of file MAIN.C.

bit Dos1TurnOn

Definition at line 85 of file MAIN.C.

bit Dos2TurnOn

Definition at line 86 of file MAIN.C.

bit Dos3TurnOn

Definition at line 87 of file MAIN.C.

tWord idata dos_livetime[3]

Definition at line 37 of file MAIN.C.

tLong idata dos_zero[3]

Definition at line 34 of file MAIN.C.

bit GSE

Definition at line 82 of file MAIN.C.

tByte idata HourCount

MCF_type idata MCF

Definition at line 80 of file MAIN.C.

bit ReadAllIsRunning

Definition at line 81 of file MAIN.C.

bit INSTRUMENT CONTROL_Enable

Definition at line 50 of file MAIN.C.

bit RunOnce

Definition at line 84 of file MAIN.C.

tLong idata RxTimer

Definition at line 60 of file MAIN.C.

Status1_type idata Status1

Definition at line 78 of file MAIN.C.

Status2_type idata Status2

Definition at line 79 of file MAIN.C.

bit TelemetryTick

Definition at line 46 of file MAIN.C.

tLong idata TickCount

Definition at line 57 of file MAIN.C.

tLong idata TotalTickCount

Definition at line 58 of file MAIN.C.

```
// Dosimeter.c : Dosimeter-related function implementations

#include "main.h"

#define CAL_PULSE_WIDTH 17

extern Status1_type idata Status1;
extern Status2_type idata Status2;
extern MCF_type idata MCF;
extern tLong idata dos[3];
extern tLong idata dos_zero[3];
extern tLong idata cal_pre[3];
extern tLong idata cal_post[3];
extern tWord idata dos_livetime[3];
extern tLong idata TickCount;
extern tLong idata TotalTickCount;
extern tByte idata HourCount;
extern bit TelemetryTick;
extern bit ADC_Start;
extern bit CalRunning;
extern bit ReadAllIsRunning;
extern bit CalMode;
extern bit RunOnce;
extern bit GSE;
extern bit INSTRUMENT CONTROL_Enable;
extern bit Dos1TurnOn;
extern bit Dos2TurnOn;
extern bit Dos3TurnOn;
extern tLong idata RxTimer;

void Init_INSTRUMENT CONTROL( void )
{
                    // 7 6 5 4   3 2 1 0
                    // -------   -------
  P1 = 0x7F;            // 0 1 1 1   1 1 1 1
                    //           +-+-+-- Dosimeters powered off
                    //       +---+-------- ADCs 1,2 disabled
                    //     +-------------- SCLK high
                    //   +---------------- MISO set for input (high)
                    // +------------------ MOSI set low

                    // 7 6 5 4   3 2 1 0
                    // -------   -------
  P3 = P3 | 0x2C;        // X X 1 X   1 1 X X
                    //              +-- RxD controlled by 8051
                    //              +---- TxD controlled by 8051
                    //            +------ CAL set high (disabled)
                    //          +-------- INT_1 set for input (high)
                    //       +----------- EEPROM_RST_CTRL - leave unchanged
                    //    +-------------- MEM_MAP set high (lower 64KB bank selected)
                    // +-+---------------- RD, WR controlled by 8051 (high)

  // We leave the "Final" bit set for every packet
  Final = 1;
}

void SysTimerSetupFast( void )
{
  // Timer 2 is to be used as a system tick timer (in the final system)
  // If we use this timer to generate a tick every 2 msec, that will
  // require a reload value that is 65536 - (0.002 seconds * Tclk).
  // For a 16.129 MHz oscillator, Finst = 16129000/12 = 1.344083 MHz
  // This results in Tinst = 744 ns. Therefore 1 ms = 1,344 instructions.
  // For a system timer interval of M msecs, we need M * 1344.
  // For 4 ms, the reload timer needs to be 65536 - 5376 = 60160.
  // In hexadecimal, this is: 0xEB00

  ET2     = 0;
  TR2     = 0;
  T2CON   = 0x04;   // Load the T2 timer control register:
                 //  T2 Run on, no mode bits to sweat
  T2MOD   = 0x00;   // Timer mode control
  TH2     = 0xEB;   // Load the upper byte
  RCAP2H  = 0xEB;   // Upper byte of the reload value
  TL2     = 0x00;   // Load the lower byte
  RCAP2L  = 0x00;   // Load the lower byte

  // Enable the T2 interrupt
  ET2     = 1;

  // Start the timer running
  TR2     = 1;
}

void SysTimerSetupSlow( void )
{
  // Timer 2 is to be used as a system tick timer (in the final system)
  // If we use this timer to generate a tick every 20 msec, that will
  // require a reload value that is 65536 - (0.02 seconds * Tclk).
  // For a 16.129 MHz oscillator, Finst = 16129000/12 = 1.344083 MHz
  // This results in Tinst = 744 ns. Therefore 1 ms = 1,344 instructions.
  // For a system timer interval of M msecs, we need M * 1344.
  // For 32ms, the reload timer needs to be 65536 - 43,008 = 22,528.
  // In hexadecimal, this is: 0x5800

  ET2     = 0;
  TR2     = 0;
  T2CON   = 0x04;   // Load the T2 timer control register:
                  //  T2 Run on, no mode bits to sweat
  T2MOD   = 0x00;   // Timer mode control
  TH2     = 0x58;   // Load the upper byte
  RCAP2H  = 0x58;   // Upper byte of the reload value
  TL2     = 0x00;   // Load the lower byte
  RCAP2L  = 0x00;   // Load the lower byte

  // Enable the T2 interrupt
  ET2     = 1;

  // Start the timer running
  TR2     = 1;
}

void SystemTick(void) interrupt INTERRUPT_T2_Overflow
{
  // Clear the T2 overflow flag
  TF2     = 0;

  // Increment the tick count
  if (!INSTRUMENT CONTROL_Enable) return;
  if (AnalysisMode == 0) {
    TickCount += 2;
    TotalTickCount += 2;
  } else {
    TickCount += 16;
    TotalTickCount += 16;
  }

  // Increment each livetime counter once every 8.7381333333 minutes.
  if (dos1_pwr) {
    if ((TotalTickCount & 0x0003FFFF) == 0) // 0000 0000 0000 0011 1111 1111 1111
1111
      dos_livetime[0]++;
  } else {
    dos_livetime[0] = 0;
  }
  if (dos2_pwr) {
    if ((TotalTickCount & 0x0003FFFF) == 0) // 0000 0000 0000 0011 1111 1111 1111
1111
      dos_livetime[1]++;
  } else {
    dos_livetime[1] = 0;
  }
  if (dos3_pwr) {
    if ((TotalTickCount & 0x0003FFFF) == 0) // 0000 0000 0000 0011 1111 1111 1111
1111
      dos_livetime[2]++;
  } else {
    dos_livetime[2] = 0;
  }

  if (CalRunning) return;
  if (ReadAllIsRunning) return;
  if (TelemetryTick) return;

  // Synchonize pins with status
  TDYN0 = dos1_pwr;
  TDYN1 = dos2_pwr;
  TDYN2 = dos3_pwr;

  // This will trigger a new ADC sampling cycle (regardless of the mode)
  ADC_Start = 1;

}

void TSK_DoCal()
{
  tWord i;
  ContentID = CAL_CONTENT;
  CalRunning = 1;

  // Transfer the most recent values (in dos[]) to cal_pre[]
  RunOnce = 0;
  read_dosimeter_all(0);
  cal_pre[0] = dos[0];
  cal_pre[1] = dos[1];
  cal_pre[2] = dos[2];

  // Run CAL pulses
  for (i = 0; i < 50000; i++) {
    pulse_cal( CAL_PULSE_WIDTH );
  }

  // Set the Cmd history to '0000', VREF Frame to "15V & 5V"
  // Clear CmdRcvd and CmdValid flags.
  Status2.value = 0x00;

  // Read the data from the three dosimeters
  RunOnce = 0;  // Want to start at the beginning of the aux ADC channels

  read_dosimeter_all(0);
  cal_post[0] = dos[0];
  cal_post[1] = dos[1];
  cal_post[2] = dos[2];

  // End of Calibration process
  CalRunning = 0;

}

/**************************************************************
* initialize_normal_mode()
* -----------------------
* This sets up the normal-mode (default) behavior
* It really is a script of several common function calls.
*
* Inputs: None
* Output: None
* Assumptions:  It is assumed that the device is being switched to
*            operate in a normal mode (at startup or from some
*            other mode)
**************************************************************/

void initialize_normal_mode( void )
{
  //  Start by initializing the normal status.
  //  Status1:  Field              Value       Bit(s)
  //            ------------------  --------    --------
  //            Sample Mode         1 (High)      7
  //            Content ID         01 (Normal)   6,5
  //            Dosimeter #1        1 (on)        4
  //            Dosimeter #2        1 (on)        3
  //            Dosimeter #3        1 (on)        2
  //            Analysis Mode       0 (Mode1)     1
  //            EEPROM Error        1 (Good)      0
  //
  Status1.value = 0xBD;

  // Get out of GSE mode
  GSE = 0;

  // Set the Cmd history to '0000', VREF Frame to "15V & 5V"
  // Clear CmdRcvd and CmdValid flags.
  Status2.value = 0x00;

  RunOnce = 0;

  // Perform initial reads of all three dosimeters
  read_dosimeter_all(0);

}

void initialize_startup_mode( void )
{
  //  Start by initializing the normal status.
  //  Status1:  Field               Value       Bit(s)
  //            ------------------  --------    --------
  //            Sample Mode         1 (High)      7
  //            Content ID         01 (Normal)   6,5
  //            Dosimeter #1        0 (off)       4
  //            Dosimeter #2        0 (off)       3
  //            Dosimeter #3        0 (off)       2
  //            Analysis Mode       0 (Mode1)     1
  //            EEPROM Error        1 (Good)      0
  //
  Status1.value = 0xA1;

  // Get out of GSE mode
  GSE = 0;

  // Set the Cmd history to '0000', VREF Frame to "15V & 5V"
  // Clear CmdRcvd and CmdValid flags.
  Status2.value = 0x00;

#ifdef __DEBUG__
  Status1.value = 0x21;
  GSE = 1;
#endif // __DEBUG__

  RunOnce = 0;
}
```

```
void ChangeToAnalysisMode2( void )
{
  // Change to the 16 ms timer interval required for Analysis Mode 2
  SysTimerSetupSlow();
  TickCount = 0;
  dos[0] = 0;
  dos[1] = 0;
  dos[2] = 0;
}

void ChangeToAnalysisMode1( void )
{
  // Change to the 2 ms timer interval required for Analysis Mode 1
  SysTimerSetupFast();
  TickCount = 0;
  dos[0] = 0;
  dos[1] = 0;
  dos[2] = 0;
}

void TSK_DoADC( void )
{
    if ( AnalysisMode == 0)
  {
    if (Dos1TurnOn) {
      dos_zero[0] = dos[0] = 0;
      Dos1TurnOn = 0;
    }
    if (Dos2TurnOn) {
      dos_zero[1] = dos[1] = 0;
      Dos2TurnOn = 0;
    }
    if (Dos3TurnOn) {
      dos_zero[2] = dos[2] = 0;
      Dos3TurnOn = 0;
    }
    read_dosimeter_low( );
  } else {
    read_dosimeter_all(1);
    if (Dos1TurnOn) {
      dos_zero[0] = dos[0];
      Dos1TurnOn = 0;
    }
    if (Dos2TurnOn) {
      dos_zero[1] = dos[1];
```

```
 Dos2TurnOn = 0;
}
if (Dos3TurnOn) {
  dos_zero[2] = dos[2];
  Dos3TurnOn = 0;
```

Interface.c File Reference
#include "main.h"

Functions

- tWord **crc16** (unsigned char *pData, unsigned char len)
- unsigned int **SLIP** (unsigned char *packet, unsigned int Packet_len)
- unsigned int **UNSLIP** (unsigned char *packet, unsigned int Packet_len)

Function Documentation

*tWord crc16 (unsigned char * pData, unsigned char len)*

Definition at line 25 of file Interface.c.

*unsigned int SLIP (unsigned char * packet, unsigned int Packet_len)*

Definition at line 84 of file Interface.c.

*unsigned int UNSLIP (unsigned char * packet, unsigned int Packet_len)*

Definition at line 122 of file Interface.c.

```
/*****************************************************

Project : INSTRUMENT CONTROL Dosimeter (Interface.c)
Version : 0.1 EU  (Engineering Unit)
Date    : 2/20/2013
Author  : Brian Zufelt
Company : COSMIAC/UNM
Comments:
This source provides all the functions to format the
output messages through the serial interface.

*****************************************************/

#include "main.h"

/*****************************************************

        Send a pointer to an array and returns a 16-bit CRC
        Check.

        Return: 16-bit CRC value

*****************************************************/

tWord crc16(unsigned char *pData, unsigned char len)
{
  unsigned int x = 0xFFFF;
  unsigned char *p = pData;
  unsigned char ch = *p;

  while (len) {
    x ^= ch;
    // Unrolling this loop results in a 27% reduction in processing time.
    if (x & 1)
      x = (x >> 1) ^ 0x8408;
    else
      x = (x >> 1);
    if (x & 1)
      x = (x >> 1) ^ 0x8408;
    else
      x = (x >> 1);
    if (x & 1)
      x = (x >> 1) ^ 0x8408;
    else
      x = (x >> 1);
    if (x & 1)
      x = (x >> 1) ^ 0x8408;
    else
      x = (x >> 1);
    if (x & 1)
      x = (x >> 1) ^ 0x8408;
    else
      x = (x >> 1);
    if (x & 1)
      x = (x >> 1) ^ 0x8408;
    else
      x = (x >> 1);
    if (x & 1)
      x = (x >> 1) ^ 0x8408;
    else
      x = (x >> 1);
    if (x & 1)
      x = (x >> 1) ^ 0x8408;
    else
      x = (x >> 1);
    len--; p++; ch = (char)*p;
  }
  return x;
}

/****************************************************************

        The SLIP Function is Encapsulates the SLIP framing to a given
        packet.

        Returns: New packet length

        NOTE: At least packet length + 2 bytes are required for start/stop
        char. Additional size requirements of the buffer depend on the occurance
        of FEND or FESC within the data packet.

****************************************************************/

unsigned int SLIP(unsigned char *packet, unsigned int Packet_len){

        int i = 0 , j=0;                                          //general
counter

        for(i=Packet_len;i>0;i--) {                    // Shift packet 1 byte to the
right
```

```
                        packet[i] = packet[i-1];
            }

            Packet_len += 2;
            //increase packet size by 2 to add new characters
            packet[0] = FEND;                                                                              //add
start char
            packet[Packet_len-1] = FEND;                                         //add end char

            for(i=(Packet_len-2);i>1;i--)           // i<1 because packet[0] has to == FEND and
packet[Packet_len] == FEND
  {
                        if(packet[i] == FEND)
    {
                                    for(j = (Packet_len); j != i; j--)
      {
                                                packet[j] = packet[j-1];
                                    }
                                    Packet_len++;
            // increase Packet length size
                                    packet[i] = FESC;
                                    packet[i+1] = TFEND;
                        }
                        else if(packet[i] == FESC)
    {
                                    for(j = (Packet_len); j != i; j--)
      {
                                                packet[j] = packet[j-1];
                                    }
                                    Packet_len++;
            // increase Packet length size
                                    packet[i] = FESC;
                                    packet[i+1] = TFESC;
                        }
  }
            return Packet_len;               //return new packet length
}

unsigned int UNSLIP( unsigned char *packet, unsigned int Packet_len )
{
  int i, j = 0;

  for (i = 0; i < Packet_len - 2; i++) {
    if (packet[i] == FESC) {
      if (packet[i+1] == TFEND) {
        packet[i] = FEND;
        for (j = i+1; j < Packet_len -1; j++) {
          packet[j] = packet[j+1];
        }
        Packet_len--;
      } else if (packet[i+1] == TFESC) {
        packet[i] = FESC;
        for (j = i+1; j < Packet_len -1; j++) {
          packet[j] = packet[j+1];
        }
        Packet_len--;
      } else {
        // Error condition! Should never reach here...
        return 0;
      }
    }
  }
  return Packet_len;
}
```

MAIN.C File Reference

#include "main.h"

Functions

- void **main** (void)

Variables

- *tLong idata **dos** [3] = 0
  INSTRUMENT CONTROL Dosimeter Telemetry Application (Main module) (Version 2.0: Flight Software)
- tLong idata **dos_zero** [3] = 0
- tLong idata **cal_pre** [3] = 0
- tLong idata **cal_post** [3] = 0
- tWord idata **dos_livetime** [3] = 0
- tWord idata **ps_15V**
- tWord idata **ps_5V**
- tWord idata **temps** [3]
- tWord idata **FrameCounter** = 0
- bit **TelemetryTick** = 0
- bit **ADC_Start** = 0
- bit **ResetImminent** = 0
- bit **TxEnable** = 0
- bit **INSTRUMENT CONTROL_Enable** = 0
- bit **CalRunning** = 0
- bit **CalMode** = 1
- tLong idata **TickCount** = 0
- tLong idata **TotalTickCount** = 0
- tWord idata **PktCnt_Cal** = 0
- tLong idata **RxTimer** = 0xFFFFFFFF
- tByte idata **XmitLength** = 0
- tByte idata **DataBuff** [48]
- tByte idata **AckBuff** [14]
- tByte idata * **pXmitPtr**
- tByte idata **RecvLength** = 6
- MessageQueue **msgQueue**
- tByte idata **RecvBuff** [10]
- tByte idata * **pRecvPtr**
- Status1_type idata **Status1**
- Status2_type idata **Status2**
- MCF_type idata **MCF**
- bit **ReadAllIsRunning** = 0
- bit **GSE** = 0
- bit **Poll** = 0

- bit **RunOnce** = 0
- bit **Dos1TurnOn** = 0
- bit **Dos2TurnOn** = 0
- bit **Dos3TurnOn** = 0
- tByte code **adc_ch_mode2** [3][5]
- tByte code **adc_ch** []

Function Documentation

void main (void)

Definition at line 108 of file MAIN.C.

Variable Documentation

tByte idata AckBuff[14]

Definition at line 65 of file MAIN.C.

tByte code adc_ch[]

```
Initial value:= { 0x03,   0x00,   0x10,   0x15,   0x03,   0x06,   0x15,
                  0x00,   0x03,   0x00,   0x11,   0x15,   0x03,   0x00,
                  0x12,   0x15,   0x03,   0x00,   0x14,   0x15 }
```

Definition at line 104 of file MAIN.C.

tByte code adc_ch_mode2[3][5]

```
Initial value:=
               {
                  { 0x00, 0x06, 0x00, 0x00, 0x00 },
                  { 0x10, 0x15, 0x11, 0x12, 0x14 },
                  { 0x15, 0x00, 0x15, 0x15, 0x15 }
               }
```

Definition at line 91 of file MAIN.C.

bit ADC_Start = 0

Definition at line 47 of file MAIN.C.

tLong idata cal_post[3] = 0

Definition at line 36 of file MAIN.C.

tLong idata cal_pre[3] = 0

Definition at line 35 of file MAIN.C.

bit CalMode = 1

Definition at line 54 of file MAIN.C.

bit CalRunning = 0

Definition at line 53 of file MAIN.C.

tByte idata DataBuff[48]

Definition at line 64 of file MAIN.C.

**** tLong idata dos[3] = 0***

INSTRUMENT CONTROL Dosimeter Telemetry Application (Main module) (Version 2.0: Flight Software)

This code implements the full INSTRUMENT CONTROL dosimeter sensor, including all message formatting, data processing, and utility functions (ADC, CRC, ...)

Version Rev. 1.0 Created from the code written by Brian Zufelt for the development board and by code written by me for the EU model.

Chip type : Aeroflex UTMC UT69RH051 Program type : Firmware Core Clock frequency : 16.129000 MHz

Definition at line 33 of file MAIN.C.

bit Dos1TurnOn = 0

Definition at line 85 of file MAIN.C.

bit Dos2TurnOn = 0

Definition at line 86 of file MAIN.C.

bit Dos3TurnOn = 0

Definition at line 87 of file MAIN.C.

tWord idata dos_livetime[3] = 0

Definition at line 37 of file MAIN.C.

tLong idata dos_zero[3] = 0

Definition at line 34 of file MAIN.C.

tWord idata FrameCounter = 0

Definition at line 43 of file MAIN.C.

bit GSE = 0

Definition at line 82 of file MAIN.C.

MCF_type idata MCF

Definition at line 80 of file MAIN.C.

MessageQueue msgQueue

Definition at line 68 of file MAIN.C.

tWord idata PktCnt_Cal = 0

Definition at line 59 of file MAIN.C.

bit Poll = 0

Definition at line 83 of file MAIN.C.

tByte idata* pRecvPtr

Definition at line 75 of file MAIN.C.

tWord idata ps_15V

Definition at line 38 of file MAIN.C.

tWord idata ps_5V

Definition at line 39 of file MAIN.C.

tByte idata* pXmitPtr

Definition at line 66 of file MAIN.C.

bit ReadAllIsRunning = 0

Definition at line 81 of file MAIN.C.

tByte idata RecvBuff[10]

Definition at line 73 of file MAIN.C.

tByte idata RecvLength = 6

Definition at line 67 of file MAIN.C.

bit ResetImminent = 0

Definition at line 48 of file MAIN.C.

bit INSTRUMENT CONTROL_Enable = 0

Definition at line 50 of file MAIN.C.

bit RunOnce = 0

Definition at line 84 of file MAIN.C.

tLong idata RxTimer = 0xFFFFFFFF

Definition at line 60 of file MAIN.C.

Status1_type idata Status1

Definition at line 78 of file MAIN.C.

Status2_type idata Status2

Definition at line 79 of file MAIN.C.

bit TelemetryTick = 0

Definition at line 46 of file MAIN.C.

tWord idata temps[3]

Definition at line 40 of file MAIN.C.

tLong idata TickCount = 0

Definition at line 57 of file MAIN.C.

tLong idata TotalTickCount = 0

Definition at line 58 of file MAIN.C.

bit TxEnable = 0

Definition at line 49 of file MAIN.C.

tByte idata XmitLength = 0

Definition at line 63 of file MAIN.C.

===

```
//!
//! \brief  INSTRUMENT CONTROL Dosimeter Telemetry Application (Main module)
//!         (Version 2.0: Flight Software)
//!
//! This code implements the full INSTRUMENT CONTROL dosimeter sensor,
//! including all message formatting, data processing, and
//! utility functions (ADC, CRC, ...)
//! ******************************************************
//! Version Rev.
//! 1.0   Created from the code written by Brian Zufelt for
//!       the development board and by code written by me
//!       for the EU model.
//! ******************************************************
//! Chip type               : Aeroflex UTMC UT69RH051
//! Program type            : Firmware
//! Core Clock frequency            : 16.129000 MHz
//! ******************************************************/
```

```
/*****************************************************

Project : INSTRUMENT CONTROL Dosimeter Telemetry Application (Main.c)
Version : 1.0 FH  (Flight Hardware)
Date    : 07/18/2013
Company : COSMIAC/UNM

*****************************************************/

#include "main.h"         // Global header

/* Global storage for measurements */
tLong idata dos[3] = 0;       // Contains the actual dosimeter values
tLong idata dos_zero[3] = 0;  // Baseline values for the three dosimeters
tLong idata cal_pre[3] = 0;  // Contains the values at power_on.
tLong idata cal_post[3] = 0; // Contains the intermediate values (dos[] + dose since last
update)
tWord idata dos_livetime[3] = 0; // Total running time for each dosimeter.
tWord idata ps_15V;           // 15V line
tWord idata ps_5V;            // 5V line
tWord idata temps[3];         // Temperatures

// Global frame counter
tWord idata FrameCounter = 0; // The frame counter - must be zero'ed out on
reset/power-up

// Command/Status
bit TelemetryTick = 0;        // This will be '1' whenever we are sending telemetry data
bit ADC_Start = 0;            // This is a flag set by the SysTimerXXX interrupt routine.
bit ResetImminent = 0;        // This will likely not be used (the "Reset Imminent" cmd is
not implemented)
bit TxEnable = 0;             // If this is clear, we continue to measure/accumulate dose, but
don't report
bit INSTRUMENT CONTROL_Enable = 0;         // This flag keeps the instrument from
starting to run until the first command

// Calibration interval counter
bit CalRunning = 0;           // '1' if the CAL pulses are being generated
bit CalMode = 1;              // CalMode = 1 --> perform weekly calibration and reset
dosimeters

// Tracking time for interrupt interval
tLong idata TickCount = 0;    // Counts 2ms intervals to trigger events
tLong idata TotalTickCount = 0;
tWord idata PktCnt_Cal = 0;
```

```
tLong idata RxTimer = 0xFFFFFFFF; // Initialize to high end of range

// Serial comm buffers
tByte idata XmitLength = 0;   // If this value is non-zero, there is data to transmit
tByte idata DataBuff[48];     // Data block containing chars to be xmit'd
tByte idata AckBuff[14];      // Data block for Ack packet chars
tByte idata *pXmitPtr;        // Pointer for iterating through the xmit buffer
tByte idata RecvLength = 6;   // Command receive length
MessageQueue msgQueue;        // Contains the pointers and lengths for data / ack
packets

#ifdef __DEBUG__
  tByte idata RecvBuff[10] = {0x56, 0x11, 0xA2, 0x7E, 0x44, 0x52, 0x00, 0x00, 0x00,
0x00};
#else
  tByte idata RecvBuff[10];
#endif // __DEBUG__
tByte idata *pRecvPtr;

// Status bytes and Command/Status flags
Status1_type idata Status1;
Status2_type idata Status2;
MCF_type idata MCF;
bit ReadAllIsRunning = 0;
bit GSE = 0;
bit Poll = 0;
bit RunOnce = 0;
bit Dos1TurnOn = 0;
bit Dos2TurnOn = 0;
bit Dos3TurnOn = 0;

// This 2D array contains the ADC dev/ch sequence for Analysis Mode 2
// (The first index is the "stage", the second is where we are in <count>)
tByte code adc_ch_mode2[3][5] =
                { //   (1)   (2)   (3)   (4)   (5)
                    { 0x00, 0x06, 0x00, 0x00, 0x00 },   // "Stage 0"
                    { 0x10, 0x15, 0x11, 0x12, 0x14 },   // "Stage 1"
                    { 0x15, 0x00, 0x15, 0x15, 0x15 }    // "Stage 2"
                };

// This 1D array contains the ADC dev/ch sequence for Analysis Mode 1
// (There are four measurements per timer tick:
//                  Dos1Low,
//                  Dos2Low,
//                  Dos3Low,
//                    one of {+15V, +5V, Temp1, Temp2, Temp3}
tByte code adc_ch[] = { 0x03,       0x00,  0x10,  0x15,  0x03,  0x06,  0x15,
                      0x00,         0x03,  0x00,  0x11,  0x15,  0x03,  0x00,
                      0x12,         0x15,  0x03,  0x00,  0x14,  0x15 };

void main(void){

  Init_INSTRUMENT CONTROL();                  // Configure pins for start-up:
  SysTimerSetupFast();                // Start the system tick timer
        com_initialize ();            // initialize interrupt driven serial I/O
        com_baudrate (4800);            // setup for 4800 baud
  pXmitPtr = (tByte *)DataBuff;   // Initialize the transmit pointer
  pRecvPtr = (tByte *)RecvBuff;   // Initialize the receive pointer
  msgQueue.nDataLength = 0;
  msgQueue.nAckLength = 0;

#ifdef __DEBUG__
  FakeData();
#endif // __DEBUG__

  // Enable the T2 interrupt, start the timer, and enable global interrupts
  ET2     = 1;
  TR2     = 1;
        EA      = 1;

  // Make sure the instrument comes up in an inactive, controlled state
  initialize_startup_mode();
  TickCount = 0;
  RunOnce = 0;

  // Debug only
#ifdef __DEBUG__
  //printf("Write this to the serial port...\r\n");
#endif // __DEBUG__

#ifdef __DEBUG__
  RecvLength = 6;  // Length of received message...
  TSK_DoCmdParse();
#endif // __DEBUG__

  while (1) {
    // This is the main execution loop that cycles forever.

    // 1. Check if it's time to create a new telemetry packet.
    TelemetryTick = TimeToXmitTelemetry();
    if (TelemetryTick == 1) {
      PktCnt_Cal++;
      if ((PktCnt_Cal >= PACKETS_PER_WEEK) && (CalMode)) {
        PktCnt_Cal = 0;
        TSK_DoCal();
        TickCount = 0;
      }
      TSK_DoPacketFormat();
    }

    // Start the ADC process
    if (ADC_Start == 1) {
      ADC_Start = 0;
      TSK_DoADC();
    }

    // 2. Check if there are characters to transmit.
    //    If so, attempt to print one character (non-blocking)
    if ((XmitLength != 0)) {
      // In the process of sending a message
      if (!com_putchar( *(pXmitPtr) )) {
        // A character was successfully placed into the buffer
        pXmitPtr++;
        XmitLength--;
      }
    } else {
      // Check for waiting telemetry or ACK packets.
      QueueWaitingMessage();
    }

    // 3. Process characters as they arrive (This is very fast!)
    TSK_ProcessRxChars();

    // End of while(1) loop ...
  }
}
```

Packet.c File Reference

#include "main.h"

Functions

- void **TSK_ProcessRxChars** (void)
- void **TSK_DoCmdParse** (void)
- void **TSK_DoAckPacket** (void)
- void **TSK_DoNakPacket** (tByte nak_code)
- void **TSK_DoPacketFormat** (void)
- char * **addChar** (char ch, const char *ptr)
- char * **addWord** (tWord val, const char *ptr)
- char * **addLong** (tLong val, const char *ptr)
- tByte **TimeToXmitTelemetry** (void)
- void **QueueWaitingMessage** (void)

Variables

- MCF_type idata **MCF**
- tLong idata **dos** [3]
  INSTRUMENT CONTROL Dosimeter Telemetry Application (Main module) (Version 2.0: Flight Software)
- tLong idata **dos_zero** [3]
- tLong idata **cal_pre** [3]
- tLong idata **cal_post** [3]
- tWord idata **dos_livetime** [3]
- tWord idata **ps_15V**
- tWord idata **ps_5V**
- tWord idata **temps** [3]
- tWord idata **FrameCounter**
- bit **CalMode**
- tLong idata **TickCount**
- tLong idata **TotalTickCount**
- bit **TelemetryTick**
- tByte idata **XmitLength**
- tByte idata **DataBuff** [48]
- tByte idata **AckBuff** [14]
- Status1_type idata **Status1**
- Status2_type idata **Status2**
- tByte idata * **pXmitPtr**
- tByte idata * **pRecvPtr**
- tByte idata **RecvLength**
- tByte idata **RecvBuff** [10]
- bit **GSE**
- bit **Poll**
- bit **TxEnable**

- bit **INSTRUMENT CONTROL_Enable**
- bit **ResetImminent**
- bit **Dos1TurnOn**
- bit **Dos2TurnOn**
- bit **Dos3TurnOn**
- tLong idata **RxTimer**
- tWord idata **PktCnt_Cal**
- MessageQueue **msgQueue**

Function Documentation

char addChar (char ch, const char * ptr)*

Definition at line 455 of file Packet.c.

char addLong (tLong val, const char * ptr)*

Definition at line 468 of file Packet.c.

char addWord (tWord val, const char * ptr)*

Definition at line 462 of file Packet.c.

void QueueWaitingMessage (void)

Definition at line 516 of file Packet.c.

tByte TimeToXmitTelemetry (void)

Definition at line 474 of file Packet.c.

void TSK_DoAckPacket (void)

Definition at line 265 of file Packet.c.

void TSK_DoCmdParse (void)

Definition at line 108 of file Packet.c.

void TSK_DoNakPacket (tByte nak_code)

Definition at line 283 of file Packet.c.

void TSK_DoPacketFormat (void)

Definition at line 303 of file Packet.c.

void TSK_ProcessRxChars (void)

Definition at line 42 of file Packet.c.

Variable Documentation

tByte idata AckBuff[14]

Definition at line 65 of file MAIN.C.

tLong idata cal_post[3]

Definition at line 36 of file MAIN.C.

tLong idata cal_pre[3]

Definition at line 35 of file MAIN.C.

bit CalMode

Definition at line 54 of file MAIN.C.

tByte idata DataBuff[48]

Definition at line 64 of file MAIN.C.

tLong idata dos[3]

INSTRUMENT CONTROL Dosimeter Telemetry Application (Main module) (Version 2.0: Flight Software)

This code implements the full INSTRUMENT CONTROL dosimeter sensor, including all message formatting, data processing, and utility functions (ADC, CRC, ...)

Version Rev. 1.0 Created from the code written by Brian Zufelt for the development board and by code written by me for the EU model.

Chip type : Aeroflex UTMC UT69RH051 Program type : Firmware Core Clock frequency : 16.129000 MHz

Definition at line 33 of file MAIN.C.

bit Dos1TurnOn

Definition at line 85 of file MAIN.C.

bit Dos2TurnOn

Definition at line 86 of file MAIN.C.

bit Dos3TurnOn

Definition at line 87 of file MAIN.C.

tWord idata dos_livetime[3]

Definition at line 37 of file MAIN.C.

tLong idata dos_zero[3]

Definition at line 34 of file MAIN.C.

tWord idata FrameCounter

Definition at line 43 of file MAIN.C.

bit GSE

Definition at line 82 of file MAIN.C.

MCF_type idata MCF

Definition at line 80 of file MAIN.C.

MessageQueue msgQueue

Definition at line 68 of file MAIN.C.

tWord idata PktCnt_Cal

Definition at line 59 of file MAIN.C.

bit Poll

Definition at line 83 of file MAIN.C.

tByte idata* pRecvPtr

Definition at line 75 of file MAIN.C.

tWord idata ps_15V

Definition at line 38 of file MAIN.C.

tWord idata ps_5V

Definition at line 39 of file MAIN.C.

tByte idata* pXmitPtr

Definition at line 66 of file MAIN.C.

tByte idata RecvBuff[10]

Definition at line 73 of file MAIN.C.

tByte idata RecvLength

Definition at line 67 of file MAIN.C.

bit ResetImminent

Definition at line 48 of file MAIN.C.

bit INSTRUMENT CONTROL_Enable

Definition at line 50 of file MAIN.C.

tLong idata RxTimer

Definition at line 60 of file MAIN.C.

Status1_type idata Status1

Definition at line 78 of file MAIN.C.

Status2_type idata Status2

Definition at line 79 of file MAIN.C.

bit TelemetryTick

Definition at line 46 of file MAIN.C.

tWord idata temps[3]

Definition at line 40 of file MAIN.C.

tLong idata TickCount

Definition at line 57 of file MAIN.C.

tLong idata TotalTickCount

Definition at line 58 of file MAIN.C.

bit TxEnable

Definition at line 49 of file MAIN.C.

tByte idata XmitLength

Definition at line 63 of file MAIN.C.

```
// Packet.c
// This file contains the code for assembling packets for transmittal to ARCS

#include "main.h"

extern MCF_type idata MCF;
extern tLong idata dos[3];
extern tLong idata dos_zero[3];
extern tLong idata cal_pre[3];
extern tLong idata cal_post[3];
extern tWord idata dos_livetime[3];
extern tWord idata ps_15V;
extern tWord idata ps_5V;
extern tWord idata temps[3];
extern tWord idata FrameCounter;
extern bit CalMode;
extern tLong idata TickCount;
extern tLong idata TotalTickCount;
extern bit TelemetryTick;
extern tByte idata XmitLength;
extern tByte idata DataBuff[48];
extern tByte idata AckBuff[14];
extern Status1_type idata Status1;
extern Status2_type idata Status2;
extern tByte idata *pXmitPtr;
extern tByte idata *pRecvPtr;
extern tByte idata RecvLength;
extern tByte idata RecvBuff[10];
extern bit GSE;
extern bit Poll;
extern bit TxEnable;
extern bit INSTRUMENT CONTROL_Enable;
extern bit ResetImminent;
extern bit Dos1TurnOn;
extern bit Dos2TurnOn;
extern bit Dos3TurnOn;
extern tLong idata RxTimer;
extern tWord idata PktCnt_Cal;
extern MessageQueue msgQueue;

// UART task functions
void TSK_ProcessRxChars(void)
{
  // Create the state variable and initialize it to starting state
  static RxStateType state = WaitingFirstFEND;
  int rx_char;
```

```
//   if ((TickCount - RxTimer) > 1000)
//   {
//     // Recover from a receive timeout - reset to idle state
//     state = WaitingFirstFEND;
//     RecvLength = 0;
//     pRecvPtr = RecvBuff;
//     RxTimer = 0xFFFFFFFF;
//     return;
//   }

  // Check comm port for a character (-1 = empty)
  rx_char = com_getchar();
  if (rx_char == -1) return;

  //com_putchar(rx_char);

  switch(state)
  {
    case WaitingFirstFEND:
      if ( ((tByte)rx_char) == FEND)
      {
        state = WaitingFirstNonFEND;
        RecvLength = 0;
        RxTimer = TickCount;
      }
      return;
      break;

    case WaitingFirstNonFEND:
      if ( (tByte)rx_char != FEND)
      {
        state = ReadingChars;
        *(pRecvPtr) = (tByte )rx_char;
        pRecvPtr++;
        RecvLength++;
      }
      return;
      break;

    case ReadingChars:
      if ( (tByte)rx_char == FEND)
      {
        state = WaitingFirstFEND;
        RxTimer = 0xFFFFFFFF;
        // Found the end of the command - now process it
        CmdFound = 1;
        TSK_DoCmdParse();
      } else {
        *(pRecvPtr) = (tByte )rx_char;
        pRecvPtr++;
        RecvLength++;
      }
      return;
      break;

    default:
      break;
  }
}

void TSK_DoCmdParse( void )
{
  tByte oldMode;
  tByte NAK_code = 0x00;

  if (AnalysisMode == 0) {
    oldMode = 0;
  } else {
    oldMode = 1;
  }

  // Ensure that the pointer is initialized to the receive buffer
  pRecvPtr = RecvBuff;

  // "Unescape" the string
  RecvLength = UNSLIP( pRecvPtr, RecvLength );

  // Check the CRC field
  pRecvPtr += (RecvLength - 2);
  if (crc16( RecvBuff, RecvLength-2 ) != *((tWord *)pRecvPtr)) {
    NAK_code = 0x03;
    goto bad_exit;
  }
  pRecvPtr = RecvBuff;  // Return the pointer to the start of the message.

  // Check the destination address
  if ( *(pRecvPtr) != INSTRUMENT CONTROL_ADDRESS) {
    NAK_code = 0x04;
    goto bad_exit;
  }
```

```
// Check the source address
pRecvPtr++;
if ( *(pRecvPtr) != ARCS_ADDRESS) {
  NAK_code = 0x05;
  goto bad_exit;
}

// Assertion: The command is from the avionics
//           host and is intended for INSTRUMENT CONTROL

// Parse the message control field
pRecvPtr++;
Poll = ( *(pRecvPtr) & 0x80 ) >> 7;
B_bit = ( *(pRecvPtr) & 0x40 ) >> 6;  // Pass through unaltered - we don't use this bit.
CmdCode = ( *(pRecvPtr) & 0x1F );
Cmd = ( *(pRecvPtr) & 0x0F );
if (Cmd == 0x01) {  // "Instrument Reset" command (1 payload byte)
  if (RecvLength == 6) {
    INSTRUMENT CONTROL_Enable = 1;
    pRecvPtr++;
    if ( *(pRecvPtr) != 0) { // Reset is imminent - perform an orderly shutdown
      ResetImminent = 1;
    } else {
      ResetImminent = 0;
    }
    goto good_exit;
  } else {
    NAK_code = 0x02;
    goto bad_exit;
  }
} else if (Cmd == 0x02) { // "Configure" command (1 payload byte)
  if (RecvLength == 6) {
    INSTRUMENT CONTROL_Enable = 1;
    CmdValid = 1;
    pRecvPtr++;
    GSE = ( *(pRecvPtr) & 0x80) >> 7;
    Status1.fields.SampleMode = ( *(pRecvPtr) & 0x40) >> 6;

    if ((!CalMode) && ((( *(pRecvPtr) & 0x20) >> 5) == 1)) {
      // If CalMode goes from 0 --> 1, force a CAL
      PktCnt_Cal = PACKETS_PER_WEEK;
      TickCount = 0xFFFF0000;
    }
    CalMode = ( *(pRecvPtr) & 0x20) >> 5;
    TxEnable = ( *(pRecvPtr) & 0x10) >> 4;
    // Test for dos1_pwr switching from off to on
    Dos1TurnOn = dos1_pwr;
    dos1_pwr = ( *(pRecvPtr) & 0x08) >> 3;
    if ((dos1_pwr) && (!Dos1TurnOn))
      Dos1TurnOn = 1;
    else
      Dos1TurnOn = 0;

    // Test for dos2_pwr switching from off to on
    Dos2TurnOn = dos2_pwr;
    dos2_pwr = ( *(pRecvPtr) & 0x04) >> 2;
    if ((dos2_pwr) && (!Dos2TurnOn))
      Dos2TurnOn = 1;
    else
      Dos2TurnOn = 0;

    // Test for dos3_pwr switching from off to on
    Dos3TurnOn = dos3_pwr;
    dos3_pwr = ( *(pRecvPtr) & 0x02) >> 1;
    if ((dos3_pwr) && (!Dos3TurnOn))
      Dos3TurnOn = 1;
    else
      Dos3TurnOn = 0;

    AnalysisMode = ( *(pRecvPtr) & 0x01);
    // Test to see if the analysis mode has changed ...
    // If it has, need to change the system timer tick interval
    if ((oldMode == 0) && (AnalysisMode == 1))
      ChangeToAnalysisMode2();
    if ((oldMode == 1) && (AnalysisMode == 0))
      ChangeToAnalysisMode1();
    goto good_exit;
  } else {
    NAK_code = 0x02;
    goto bad_exit;
  }
} else if (Cmd == 0x03) { // "Normal mode" command (0 payload bytes)
  if (RecvLength == 5) {
    // Initialize the normal mode of operation
    INSTRUMENT CONTROL_Enable = 1;
    // Set the TurnOn flag for each dosimeter that was off
    if (!dos1_pwr) Dos1TurnOn = 1;
    if (!dos2_pwr) Dos2TurnOn = 1;
    if (!dos3_pwr) Dos3TurnOn = 1;
    initialize_normal_mode();
    // Change to faster timer tick
```

```
      if (oldMode == 1)
        ChangeToAnalysisMode1();
      goto good_exit;
    } else {
      NAK_code = 0x02;
      goto bad_exit;
    }
  }
  else {
    NAK_code = 0x01;
    goto bad_exit;
  }

  // Don't often use GOTOs, but sometimes they are useful
bad_exit:
  CmdValid = 0;
  Ack = 0;
  if (Poll)
    TSK_DoNakPacket( NAK_code );
  goto exit_func;

good_exit:
  CmdValid = 1;
  Ack = 1;
  if (Poll)
    TSK_DoAckPacket();
  Ack = 0;

exit_func:
  pRecvPtr = (tByte *)RecvBuff;
  return;
}

// ACK/NAK packet formatting functions
void TSK_DoAckPacket( void )
{
  // Start the data insertion point just past the MCF
  char *ptr = AckBuff + 3;
  tWord crc;

  // Pre-populate the values that are always the same
  AckBuff[0] = 0x11;       // Destination address (host(ARCS) = 0x11)
  AckBuff[1] = 0x56;       // Source address (INSTRUMENT CONTROL = 0x56)
  AckBuff[2] = MCF.value;  // Message Control Field =
                           // 0xA0 for most NSP telemetry packets
  crc = crc16( (char *)(AckBuff), 3);
  ptr = addWord( crc, ptr);                // Bytes 4,5

  msgQueue.nAckLength = SLIP( AckBuff, ptr - AckBuff);
  return ;
}

void TSK_DoNakPacket( tByte nak_code )
{
  // Start the data insertion point just past the MCF
  char *ptr = AckBuff + 4;
  tWord crc;

  // Pre-populate the values that are always the same
  AckBuff[0] = 0x11;       // Destination address (host(ARCS) = 0x11)
  AckBuff[1] = 0x56;       // Source address (INSTRUMENT CONTROL = 0x56)
  AckBuff[2] = MCF.value;  // Message Control Field =
                           // 0xA0 for most NSP telemetry packets
  AckBuff[3] = nak_code;
  crc = crc16( (char *)(AckBuff), 4);
  ptr = addWord( crc, ptr);                // Bytes 5,6

  msgQueue.nAckLength = SLIP( AckBuff, ptr - AckBuff);
  return ;
}

// Telemetry/Calibration packet formatting function
void TSK_DoPacketFormat( void )
{
  tLong temp;
  // Start the data insertion point just past the MCF
  char *ptr = DataBuff + 3;

  // There are six non-variable characters in the packet
  // They are:  Status1, Status2, and the subcommutated
  //          VREF Frame data
  // NOTE:  This is used for determining the length of the
  //       data/status string for CRC calculation
  tByte data_length = 6;

  tWord crc;
  CmdCode = 0;

  // Pre-populate the values that are always the same
  DataBuff[0] = 0x11;       // Destination address (host(ARCS) = 0x11)
  DataBuff[1] = 0x56;       // Source address (INSTRUMENT CONTROL = 0x56)
  DataBuff[2] = MCF.value;  // Message Control Field =
                           // 0xA0 for most NSP telemetry packets

  // Reset the "ACK" bit if set
  Ack = 0;

  // Insert the frame counter
  ptr = addWord( FrameCounter, ptr );

  // Insert the two status bytes
  ptr = addChar( Status1.value, ptr );
  ptr = addChar( Status2.value, ptr );

  // Insert the (active) dosimeter channels
  if (dos1_pwr) {  // Dosimeter #1 active
    if (ContentID == CAL_CONTENT)
      temp = cal_post[0] - cal_pre[0];
    else
      temp = dos[0] - dos_zero[0];
    if (temp > 0x80000000) temp = 0;
    ptr = addLong( temp, ptr );
    data_length += 4;
  }
  if (dos2_pwr) {  // Dosimeter #2 active
    if (ContentID == CAL_CONTENT)
      temp = cal_post[1] - cal_pre[1];
    else
      temp = dos[1] - dos_zero[1];
    if (temp > 0x80000000) temp = 0;
    ptr = addLong( temp, ptr );
    data_length += 4;
  }
  if (dos3_pwr) {  // Dosimeter #3 active
    if (ContentID == CAL_CONTENT)
      temp = cal_post[2] - cal_pre[2];
    else
      temp = dos[2] - dos_zero[2];
    if (temp > 0x80000000) temp = 0;
    ptr = addLong( temp, ptr );
    data_length += 4;
  }

  // Reset the flag that indicates that we are in the process of
  // formatting the buffer string (this prevents ADC sampling during
  // this period).  NOTE: NEED TO CHECK WITH CHAD ON THIS.
  // Note: This time should be very short, since most of the time
  // spent in this whole routine is spent later in the CRC calcs.
```

```
TelemetryTick = 0;
if (ContentID == CAL_CONTENT) {
  VrefFrame = 0;
}

// Insert the subcommutated power supply and temperature readings
if ( VrefFrame == 0x00 ) {         // VREF Frame: 15V & 5V
  ptr = addWord( ps_15V, ptr );
  ptr = addWord( ps_5V, ptr );
} else if( VrefFrame == 0x01 ) {   // VREF Frame: Temp1 & Temp2
  ptr = addWord( temps[0], ptr );
  ptr = addWord( temps[1], ptr );
} else if( VrefFrame == 0x02 ) {   // VREF Frame: Temp3 & DOS1 run time
  ptr = addWord( temps[2], ptr );
  ptr = addWord( dos_livetime[0], ptr );
} else {
  ptr = addWord( dos_livetime[1], ptr );  // VREF Frame: DOS2/3 run time
  ptr = addWord( dos_livetime[2], ptr );  // VREF Frame: DOS2/3 run time
}

// Insert the calculated CRC for the data packet
// Note: The fields included in this calculation are
//       Status 1        1
//       Status 2        1
//       Dosimeter 1     0, 4
//       Dosimeter 2     0, 4
//       Dosimeter 3     0, 4
//       VREF Frame      4
//                 ------
//       Total Length:   {6, 10, 14, or 18}
//
crc = crc16( (char *)(DataBuff+5), data_length);
ptr = addWord( crc, ptr);

// Insert the message CRC
// Note: The fields included in this calculation are
//       Dest address    1
//       Src Address     1
//       Msg Ctrl Field  1
//       Frame Counter   2
//       Status 1        1
//       Status 2        1
//       Dosimeter 1     0, 4
//       Dosimeter 2     0, 4
//       Dosimeter 3     0, 4
//       VREF Frame      4
```

```
  //       Payload CRC     2
  //                 ------
  //       Total Length:   {13, 17, 21, or 25}
  //
  crc = crc16( (char *)(DataBuff), data_length + 7);
  ptr = addWord( crc, ptr);

  // Set UART send data pointer to the start of the message
  pXmitPtr = DataBuff;

  FrameCounter++;
  // Increment the frame counter and advance the VREF Frame (in Status2)
  // Only want to do hourly dump or CAL once then return to normal

  /*  Hourly updates are not used at this point - we are sending total dose
      values each packet now...
  if (ContentID == HOUR_CONTENT) ContentID = NORM_CONTENT;
  */
  if (ContentID == CAL_CONTENT)
  {
    ContentID = NORM_CONTENT;
    VrefFrame = 0;
  } else {
    VrefFrame++;
    //if (VrefFrame == 3) VrefFrame = 0;
  }

  // Reset CmdFound and CmdValid bits
  CmdFound = 0;
  CmdValid = 0;

  // 'Escape' any FEND or FESC characters in the data/validation packet
  // and, if transmit is enabled, return the result (the length of the
  // SLIP'ed data frame). If transmit is disabled, returning a 0 will not
  // cause any characters to be sent.
  if (TxEnable) {
    msgQueue.nDataLength = SLIP( DataBuff, ptr - DataBuff);
  }
}

char *addChar( char ch, const char *ptr )
{
  char *p = ptr;
  *(p) = ch;
  return p+1;
}
```

```
char *addWord( tWord val, const char *ptr )
{
 // Need to add a word to the data, but safely (using addChar() method)
 return addChar( (char)(val >> 8), addChar( (char)(val & 0x00FF), ptr));
}

char *addLong( tLong val, const char *ptr )
{
 // Need to add a long (4 bytes) to the data, but safely (using addWord() method)
 return addWord( (tWord)(val >> 16), addWord( (tWord )(val & 0x0000FFFF), ptr));
}

tByte TimeToXmitTelemetry( void )
{
 // Math:  With a tick timer of 2 ms, there are 500 interrupts
 //        per second. Therefore, we multiply the number of
 //        seconds in the interval by 500 and convert it to hex.
 if (Status1.fields.SampleMode == 0) {
  // Low-speed (every 10 minutes)
  if (GSE == 1) {
   if (TickCount >= (tLong )0x000009C4) { // 2,500  (5 secs)
    TickCount = 0;
    return 1;
   } else {
    return 0;
   }
  } else {
   if (TickCount >= (tLong)0x000493E0) { // 300,000 (10 mins)
    TickCount = 0;
    return 1;
   } else {
    return 0;
   }
  }
 } else {
  // High-speed (every 2 minutes)
  if (GSE == 1) {
   if (TickCount >= (tLong )0x000000FA) { // 250   (0.5 secs)
    TickCount = 0;
    return 1;
   } else {
    return 0;
   }
  } else {
   if (TickCount >= (tLong )0x0000EA60) { // 60,000  (2 mins)
    TickCount = 0;
```

```
        return 1;
      } else {
        return 0;
      }
    }
  }
}

void QueueWaitingMessage(void)
{
  if (msgQueue.nDataLength != 0) {
    pXmitPtr = DataBuff;
    XmitLength = msgQueue.nDataLength;
    msgQueue.nDataLength = 0;
  } else if (msgQueue.nAckLength != 0) {
    pXmitPtr = AckBuff;
    XmitLength = msgQueue.nAckLength;
    msgQueue.nAckLength = 0;
  }
}

#ifdef __DEBUG__
void FakeData( void )
{
  FrameCounter = 0x0000;
  Status1.value = 0x55;
  Status2.value = 0xE1;
  dos1_pwr = 1;
  dos2_pwr = 1;
  dos3_pwr = 1;
  dos[0] = 0x00000000;
  dos[1] = 0x00000000;
  dos[2] = 0x00000000;
  ps_15V = 0x7FC0;
  ps_5V = 0x7FDB;
  temps[0] = 3115;
  temps[1] = 3002;
  temps[2] = 2975;
}
#endif // __DEBUG__
```

SIO.C File Reference

#include "main.h"

Macros

- #define **TBUF_SIZE** 16 /*** Must be one of these powers of 2 (2,4,8,16,32,64,128) ***/
- #define **RBUF_SIZE** 16 /*** Must be one of these powers of 2 (2,4,8,16,32,64,128) ***/
- #define **TBUF_SPACE** idata /*** Memory space where the transmit buffer resides ***/
- #define **RBUF_SPACE** idata /*** Memory space where the receive buffer resides ***/
- #define **CTRL_SPACE** data /*** Memory space for the buffer indexes ***/

Macro Definition Documentation

*#define CTRL_SPACE data /*** Memory space for the buffer indexes ***/*

Definition at line 26 of file SIO.C.

*#define RBUF_SIZE 16 /*** Must be one of these powers of 2 (2,4,8,16,32,64,128) ***/*

Definition at line 21 of file SIO.C.

*#define RBUF_SPACE idata /*** Memory space where the receive buffer resides ***/*

Definition at line 24 of file SIO.C.

*#define TBUF_SIZE 16 /*** Must be one of these powers of 2 (2,4,8,16,32,64,128) ***/*

Definition at line 20 of file SIO.C.

*#define TBUF_SPACE idata /*** Memory space where the transmit buffer resides ***/*

Definition at line 23 of file SIO.C.

```
/*-----------------------------------------------------------------------------
SIO.C:  Serial Communication Routines.

Copyright 1995-2002 KEIL Software, Inc.
-----------------------------------------------------------------------------*/

#include "main.h"

/*-----------------------------------------------------------------------------
Notes:

The length of the receive and transmit buffers must be a power of 2.

Each buffer has a next_in and a next_out index.

If next_in = next_out, the buffer is empty.

(next_in - next_out) % buffer_size = the number of characters in the buffer.
-----------------------------------------------------------------------------*/
#define TBUF_SIZE   16          /*** Must be one of these powers of 2
(2,4,8,16,32,64,128) ***/
#define RBUF_SIZE   16          /*** Must be one of these powers of 2
(2,4,8,16,32,64,128) ***/

#define TBUF_SPACE  idata       /*** Memory space where the transmit buffer resides
***/
#define RBUF_SPACE  idata       /*** Memory space where the receive buffer resides
***/

#define CTRL_SPACE  data        /*** Memory space for the buffer indexes ***/

#define INTERRUPT_UART 4        /*** Interrupt level for Serial Port

/*-----------------------------------------------------------------------------
-----------------------------------------------------------------------------*/
#if TBUF_SIZE < 2
#error TBUF_SIZE is too small.  It must be larger than 1.
#elif TBUF_SIZE > 128
#error TBUF_SIZE is too large.  It must be smaller than 129.
#elif ((TBUF_SIZE & (TBUF_SIZE-1)) != 0)
#error TBUF_SIZE must be a power of 2.
#endif

#if RBUF_SIZE < 2
#error RBUF_SIZE is too small.  It must be larger than 1.
#elif RBUF_SIZE > 128
```

```
#error RBUF_SIZE is too large.  It must be smaller than 129.
#elif ((RBUF_SIZE & (RBUF_SIZE-1)) != 0)
#error RBUF_SIZE must be a power of 2.
#endif

/*---------------------------------------------------------------------------
---------------------------------------------------------------------------*/
static TBUF_SPACE unsigned char tbuf [TBUF_SIZE];
static RBUF_SPACE unsigned char rbuf [RBUF_SIZE];

static CTRL_SPACE unsigned char t_in = 0;
static CTRL_SPACE unsigned char t_out = 0;

static CTRL_SPACE unsigned char r_in = 0;
static CTRL_SPACE unsigned char r_out = 0;

static bit ti_restart = 0;  /* NZ if TI=1 is required */


/*---------------------------------------------------------------------------
---------------------------------------------------------------------------*/
static void com_isr (void) interrupt INTERRUPT_UART
{
/*------------------------------------------------
Received data interrupt.
------------------------------------------------*/
if (RI != 0)
  {
  RI = 0;

  if (((r_in - r_out) & ~(RBUF_SIZE-1)) == 0)
    {
    rbuf [r_in & (RBUF_SIZE-1)] = SBUF;
    r_in++;
    }
  }

/*------------------------------------------------
Transmitted data interrupt.
------------------------------------------------*/
if (TI != 0)
  {
  TI = 0;

  if (t_in != t_out)
    {
```

```
    SBUF = tbuf [t_out & (TBUF_SIZE-1)];
    t_out++;
    ti_restart = 0;
    }
  else
    {
    ti_restart = 1;
    }
  }

}

/*---------------------------------------------------------------------------
---------------------------------------------------------------------------*/
#pragma disable

void com_initialize (void)
{
/*----------------------------------------------
Setup TIMER1 to generate the proper baud rate.
----------------------------------------------*/
com_baudrate (1200);                              // Was 1200

/*----------------------------------------------
Clear com buffer indexes.
----------------------------------------------*/
t_in = 0;
t_out = 0;

r_in = 0;
r_out = 0;

/*----------------------------------------------
Setup serial port registers.
----------------------------------------------*/
SM0 = 0; SM1 = 1;   /* serial port MODE 1 */
SM2 = 0;
REN = 1;            /* enable serial receiver */

RI = 0;             /* clear receiver interrupt */
TI = 0;             /* clear transmit interrupt */
ti_restart = 1;

ES = 1;             /* enable serial interrupts */
PS = 0;             /* set serial interrupts to low priority */
}
```

```
/*---------------------------------------------------------------------------
---------------------------------------------------------------------------*/
#pragma disable

void com_baudrate (
  unsigned baudrate)
{
/*----------------------------------------------
Clear transmit interrupt and buffer.
----------------------------------------------*/
TI = 0;             /* clear transmit interrupt */
t_in = 0;           /* empty transmit buffer */
t_out = 0;

/*----------------------------------------------
Set timer 1 up as a baud rate generator.
----------------------------------------------*/
TR1 = 0;            /* stop timer 1 */
ET1 = 0;            /* disable timer 1 interrupt */

PCON |= 0x80;       /* 0x80=SMOD: set serial baudrate doubler */

TMOD &= ~0xF0;      /* clear timer 1 mode bits */
TMOD |= 0x20;       /* put timer 1 into MODE 2 */

TH1 = (unsigned char) (256 - (16129000 / (16L * 12L * baudrate)));

TR1 = 1;            /* start timer 1 */
}

/*---------------------------------------------------------------------------
---------------------------------------------------------------------------*/
#pragma disable

char com_putchar (
  unsigned char c)
{
/*----------------------------------------------
If the buffer is full, return an error value.
----------------------------------------------*/
if (com_tbuflen () >= TBUF_SIZE)
  return (-1);

/*----------------------------------------------
Add the data to the transmit buffer.  If the
transmit interrupt is disabled, then enable it.
-----------------------------------------------*/
tbuf [t_in & (TBUF_SIZE - 1)] = c;
t_in++;

if (ti_restart)
  {
  ti_restart = 0;
  TI = 1;               /* generate transmit interrupt */
  }

return (0);
}

/*---------------------------------------------------------------------------
---------------------------------------------------------------------------*/
#pragma disable

int com_getchar (void)
{
if (com_rbuflen () == 0)
  return (-1);

return (rbuf [(r_out++) & (RBUF_SIZE - 1)]);
}

/*---------------------------------------------------------------------------
---------------------------------------------------------------------------*/
#pragma disable

unsigned char com_rbuflen (void)
{
return (r_in - r_out);
}

/*---------------------------------------------------------------------------
---------------------------------------------------------------------------*/
#pragma disable

unsigned char com_tbuflen (void)
{
return (t_in - t_out);
}
```

List of Abbreviations, Symbols, and Acronyms

AFRL – Air Force Research Laboratory

COSMIAC – Configurable Space Microsystems Innovations and Applications Center

EEPROM – Electrically Erasable Programmable Read-Only Memory

LEO – Low Earth Orbit

MIT – Massachusetts Institute of Technology

NASA – National Aeronautics and Space Administration

RHAS - Radiation Hazard Assessment System

SPA – Space Plug-and-play Architecture

SDM – Satellite Data Module

SSM – Satellite System Module

UNM – University of New Mexico

UNP – University Nanosat Program

DISTRIBUTION LIST

DTIC/OCP 8725 John J. Kingman Rd, Suite 0944 Ft Belvoir, VA 22060-6218	1 cy
AFRL/RVIL Kirtland AFB, NM 87117-5776	2 cys
Official Record Copy AFRL/RVSE/Keith Avery	1 cy